Middle East Technical University | Department of Computer Engineering

CENG 213

Data Structures

Fall 2019-2020

Homework 3

Due date: 20.12.2019, Friday, 23:55

1 Introduction

In this assignment, you are going to practice on hash table implementation for a book database, along with its accompanying hash function using C++. Remember that insertion, deletion, and retrieval operations are to run in expected constant time for hash tables.

Keywords: *Hash table, Quadratic hashing, C++*

2 Problem Definition

In this homework, you will design a hash table for a book database. The application must store books (have some attributes) in the hash table according to their isbn. Note that, each book has a unique isbn. Your application need to provide the following requirements:

- Inserting a book into the hash table
- Deleting a book from the hash table
- Getting the book with the given isbn

3 Specifications

- In this hash table, for every possible key in the table, you are supposed to store at most 3 values, called a bucket. If the bucket for a key is used, then your hash table should use the standard open addressing (quadratic probing) to find the next bucket for that key. Note that in a bucket, you should search the entries linearly, don't iterate in the bucket quadratically.
  Example

```
// assume Hash function is i % 10

HashTable<int> table;
table.Insert(3, 11111);
table.Insert(13, 11112);
table.Insert(23, 11113);


table.Insert(33, 11114);
```

Note that, this example is for illustration purpose. In the assignment, all keys will be string and values will be a generic class. All (3, 11111) , (13, 11112) , (23, 11113) entries should be stored in the same bucket. However, (33, 11114) entry must be stored in another bucket since the bucket of the key 3 is full now:

Bucket	Entry 0	Entry 1	Entry 2
0			
1			
2			
3	(3, 11111)	(13, 11112)	(23, 11113)
4	(33, 11114)		

- You will be implementing a hash table class, named as HashTable. The bare header file, "HashTable.h", is given below. You are free to add any private variables/functions to it. However, your hash table should support at least the given functions.

```
template <class T>
class HashTable {
    struct Entry {
        std::string Key;              // the key of the entry
        T Value;   // the value of the entry
        bool Deleted;         // flag indicating whether this entry is deleted
        bool Active;          // flag indicating whether this item is currently used

        Entry() : Key(), Value(), Deleted(false), Active(false) {}
    };

    struct Bucket {
        Entry entries[3];
    };

    int _capacity; // INDICATES THE SIZE OF THE TABLE
    int _size; // INDICATES THE NUMBER OF ITEMS IN THE TABLE

    Bucket* _table; // HASH TABLE

    // You can define private methods and variables

public:
    // TODO: IMPLEMENT THESE FUNCTIONS.
    // CONSTRUCTORS, ASSIGNMENT OPERATOR, AND THE DESTRUCTOR
    HashTable();
    HashTable(const HashTable<T>& rhs);
    HashTable<T>& operator=(const HashTable<T>& rhs);
    ~HashTable();

    // TODO: IMPLEMENT THIS FUNCTION.
    // INSERT THE ENTRY IN THE HASH TABLE WITH THE GIVEN KEY & VALUE
    // IF THE GIVEN KEY ALREADY EXISTS, THE NEW VALUE OVERWRITES
    // THE ALREADY EXISTING ONE.
    // IF LOAD FACTOR OF THE TABLE IS BIGGER THAN 0.5,
    // RESIZE THE TABLE WITH THE NEXT PRIME NUMBER.
    void Insert(std::string key, const T& value);

    // TODO: IMPLEMENT THIS FUNCTION.
    // DELETE THE ENTRY WITH THE GIVEN KEY FROM THE TABLE
    // IF THE GIVEN KEY DOES NOT EXIST IN THE TABLE, JUST RETURN FROM THE FUNCTION
    // HINT: YOU SHOULD UPDATE ACTIVE & DELETED FIELDS OF THE DELETED ENTRY.
    void Delete(std::string key);

    // TODO: IMPLEMENT THIS FUNCTION.
    // IT SHOULD RETURN THE VALUE THAT CORRESPONDS TO THE GIVEN KEY.
    // IF THE KEY DOES NOT EXIST, THIS FUNCTION MUST RETURN T()
    T Get(std::string key) const;

    // TODO: IMPLEMENT THIS FUNCTION.
    // AFTER THIS FUNCTION IS EXECUTED THE TABLE CAPACITY MUST BE
    // EQUAL TO newCapacity AND ALL THE EXISTING ITEMS MUST BE REHASHED
    // ACCORDING TO THIS NEW CAPACITY.
    // WHEN CHANGING THE SIZE, YOU MUST REHASH ALL OF THE ENTRIES FROM 0TH ENTRY TO LAST ↩
        ENTRY
    void Resize(int newCapacity);

    // TODO: IMPLEMENT THIS FUNCTION.
```

```
    // RETURNS THE AVERAGE NUMBER OF PROBES FOR SUCCESSFUL SEARCH
    double getAvgSuccessfulProbe();

    // TODO: IMPLEMENT THIS FUNCTION.
    // RETURNS THE AVERAGE NUMBER OF PROBES FOR UNSUCCESSFUL SEARCH
    double getAvgUnsuccessfulProbe();

    // THE IMPLEMENTATION OF THESE FUNCTIONS ARE GIVEN TO YOU
    // DO NOT MODIFY!
    int Capacity() const;
    int Size() const;
};
```

- A book's attributes will be represented with the Book class. It should not be modified in any case.

```
class Book {

private:
    string isbn;
    string name;
    string category;
    string writer;
    string publisher;
    int first_pub_date;
    int page_count;

public:
// Constructors
    Book() : isbn("") {}

    Book(const string isbn, const string name, const string category, const string writer,
      const string publisher,int first_pub_date, int page_count) : isbn(isbn), name(name),
         category(category), writer(writer),publisher(publisher),
         first_pub_date(first_pub_date), page_count(page_count){}

// Getters and setters
    const string &getIsbn() const {
        return isbn;
    }

    void setIsbn(const string &isbn) {
        Book::isbn = isbn;
    }

    const string &getName() const {
        return name;
    }

    void setName(const string &name) {
        Book::name = name;
    }

    const string &getCategory() const {
        return category;
    }

    void setCategory(const string &category) {
        Book::category = category;
    }

    const string &getWriter() const {
        return writer;
    }

    void setWriter(const string &writer) {
        Book::writer = writer;
    }

    const string &getPublisher() const {
        return publisher;
    }

    void setPublisher(const string &publisher) {
```

```
        Book::publisher = publisher;
    }

    int getFirst_pub_date() const {
        return first_pub_date;
    }

    void setFirst_pub_date(int first_pub_date) {
        Book::first_pub_date = first_pub_date;
    }

    int getPage_count() const {
        return page_count;
    }

    void setPage_count(int page_count) {
        Book::page_count = page_count;
    }
};
```

- You are given a utility class, "HashUtils" which contains a hash function and a function for finding the next appropriate size for the hash table. It should not be modified in any case.

```
#ifndef __HASHUTILS__
#define __HASHUTILS__
#include <string>
/*
* Returns the hash value of the given key
*/
int Hash(std::string key);

/*
* Finds the next appropriate hash table size from a given number
*/
int NextCapacity(int prev);

#endif
```

- You will implement functions (marked with "TODO") in the "HashTable.h", and must use open addressing with **quadratic probing** as a collision resolution strategy.
- You can assume that isbn of each book will be unique.
- A sample dataset was given to you, which includes name, category, writer, publisher, first_pub_date and page_count as well as their isbn. Test datasets will resemble the given dataset, however, they will also include new books.
- You can define new functions, variables etc. to the HashTable class. However you are not allowed to modify or remove any of the given functions or variables.
- In the default constructor, you can assume that size and capacity of the table is zero.
- **You have to call hash method in the "HashUtils.h" to calculate the hash value of an isbn.**
- **"NextCapacity" function takes an unsigned integer and returns a bigger prime number. You have to call this method while resizing the table.**
- While calculating the total probe, do not consider traversal within a bucket. In other words, you should just take account of the number of quadratic probing while calculating the total number of probe.
- While calculating the load factor, you should use 3∗_capacity as the size of the hash table.

3.1 Entry Struct

- Each "Entry" object stores information regarding a single key-value pair in the hash table.
- "Key" variable should store the original key variable given in the "Insert" method of the entry
- "Value" variables stores the value given in the "Insert" method.
- "Active" variable denotes that the key-value pair stored in this entry is valid. For example, 0^{th} entry of the 1^{st} bucket in the example above is not valid, however 0th entries of the 3^{rd} and 4^{th} bucket are valid and have their "Active" as true. Initially, "Active" variable of each "Entry" object is set to false.
- "Deleted" variable stores whether this entry has been deleted before. Initially, every "Entry" object has this variable as false.

4 Dataset

You will be provided a book dataset where fields are seperated by | character. For the scope of this assignment you are given a function to read the books from the file. The file format is given below:

```
isbn|name|category|writer|publisher|first_pub_date|page_count
9786051853154|Son|Roman|Ayse Kulin|Everest Yayinlari|2018|304
...
```

5 Regulations

1. You will use C++.
2. You are not allowed to use vector or any other standard libraries.
3. You are not allowed to modify already implemented functions and variables.
4. You can define private methods and variables if necessary.
5. If your submission fails to follow the specifications or does not compile, there will be a "significant" penalty of grade reduction.
6. Those who do the operations (insert, delete, resize, get) without utilizing the hashing will receive 0 grade.
7. Those who modify already implemented functions and those who insert other data variables or public functions and those who change the prototype of given functions will receive 0 grade.
8. **Late Submission:** Late submission policy is stated in the course syllabus.
9. **Cheating:** We have zero tolerance policy for cheating. In case of cheating, all parts involved (source(s) and receiver(s)) get zero. People involved in cheating will be punished according to the university regulations. Note that students of this course are bounded to code of honor and its violation is subject to severe punishment.
10. **Newsgroup:** You must follow the Forum (in Odtuclass) for discussions and possible updates on a daily basis.

6 Submission

1. Submission will be done via Moodle (cengclass.ceng.metu.edu.tr).
2. Do not submit a main function in any of your source files.
3. A test environment will be ready in Moodle.
   - You can submit your source files to Moodle and test your work with a subset of evaluation inputs and outputs.
   - Additional test cases will be used for evaluation of your final grade. So, your actual grades may be different than the ones you get in Moodle.
   - Only the last submission before the deadline will be graded.